LOGIC INSTRUCTIONS AND PROGRAMS

Outlines

- Define the truth tables for logic functions AND, OR, XOR
- Code 8051 Assembly language logic function instructions
- Use 8051 logic instructions for bit manipulation
- Use compare and jump instructions for program control
- Code 8051 rotate and swap instructions
- Code 8051 programs for ASCII and BCD data conversion

AND

ANL destination, source ;dest = dest AND source

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Example 7-1

Show the results of the following.

```
MOV   A,#35H    ;A = 35H
ANL   A,#0FH    ;A = A AND 0FH (now A = 05)
```

Solution:

```
35H   0 0 1 1 0 1 0 1
0FH   0 0 0 0 1 1 1 1
05H   0 0 0 0 0 1 0 1       35H AND 0FH = 05H
```

OR

ORL destination, source ;dest = dest OR source

X	Y	X AND Y
0	0	0
0	1	1
1	0	1
1	1	1

Example 7-2

Show the results of the following.

```
MOV  A,#04          ;A = 04
ORL  A,#68H         ;A = 6C
```

Solution:

```
04H      0000 0100
68H      0110 1000
6CH      0110 1100      04 OR 68 = 6CH
```

XOR

XRL destination, source ;dest = dest XOR source

X	Y	X AND Y
0	0	0
0	1	1
1	0	1
1	1	0

Example 7-3

Show the results of the following.

```
MOV     A,#54H
XRL     A,#78H
```

Solution:

```
54H          0 1 0 1 0 1 0 0
78H          0 1 1 1 1 0 0 0
2CH          0 0 1 0 1 1 0 0    54H XOR 78H = 2CH
```

```
XRL     A,#04H  ;EX-OR A with 0000 0100
```

XOR

Example 7-4

The XRL instruction can be used to clear the contents of a register by XORing it with itself. Show how "`XRL A,A`" clears A, assuming that AH = 45H.

Solution:

```
45H     01000101
45H     01000101
00      00000000 XOR a number with itself = 0
```

CPL (complement accumulator)

```
MOV A,#55H
CPL       A          ;now  A=AAH
                     ;0101 0101(55H) becomes
                     ;1010 1010 (AAH)
```

Example 7-6

Find the 2's complement of the value 85H.

Solution:

```
        MOV   A,#85H                  85H = 1000 0101
        CPL   A          ;1's comp.   1'S = 0111 1010
        ADD   A,#1       ;2's comp.              + 1
                                      0111 1011 = 7BH
```

Compare instruction

CJNE destination, source ,relative address

Example 7-7

Examine the following code, then answer the following questions.
(a) Will it jump to NEXT?
(b) What is in A after the CJNE instruction is executed?

```
        MOV   A,#55H
        CJNE  A,#99H,NEXT
        ...
NEXT:   ...
```

Solution:

(a) Yes, it jumps because 55H and 99H are not equal.
(b) A = 55H, its original value before the comparison.

Table 7-1:Carry Flag Setting For CJNE Instruction	
Compare	**Carry Flag**
destination > source	**CY = 0**
destination < source	**CY = 1**

```
                CJNE    R5,#80,NOT_EQUAL    ;check R5 for 80
                ....                        ;R5=80
NOT_EQUAL:      JNC     NEXT                ;jump if R5>80
                ....                        ;R5<80
NEXT:           ....
```

Example 7-8

Write code to determine if register A contains the value 99H. If so, make R1 = FFH; otherwise, make R1 = 0.

Solution:

```
      MOV   R1,#0             ;clear R1
      CJNE  A,#99H,NEXT       ;if A not equal 99, then jump
      MOV   R1,#0FFH          ;they are equal, make R1=FFH
NEXT:...                      ;not equal so R1=0
OVER:...
```

Example 7-9

Assume that P1 is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75	then A = 75
If T < 75	then R1 = T
If T > 75	then R2 = T

Solution:

```
        MOV   P1,#0FFH        ;make P1 an input port
        MOV   A,P1            ;read P1 port, temperature
        CJNE  A,#75,OVER      ;jump if A not equal to 75
        SJMP  EXIT            ;A=75, exit
OVER:   JNC   NEXT            ;if CY=0 then A>75
        MOV   R1,A            ;CY=1, A<75, save in R1
        SJMP  EXIT            ; and exit
NEXT:   MOV   R2,A            ;A>75, save it in R2
EXIT:   ...
```

Example 7-10

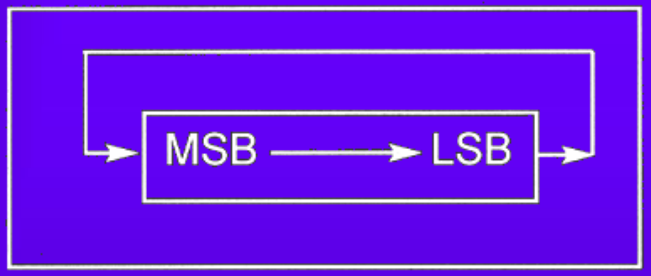
Write a program to monitor P1 continuously for the value 63H. It should get out of the monitoring only if P1 = 63H.

Solution:

```
      MOV P1,#0FFH          ;make P1 an input port
HERE: MOV A,P1              ;get P1
      CJNE A,#63,HERE       ;keep monitoring unless P1=63H
```

Rotating the bits of A right and left

RR A ;rotate right A



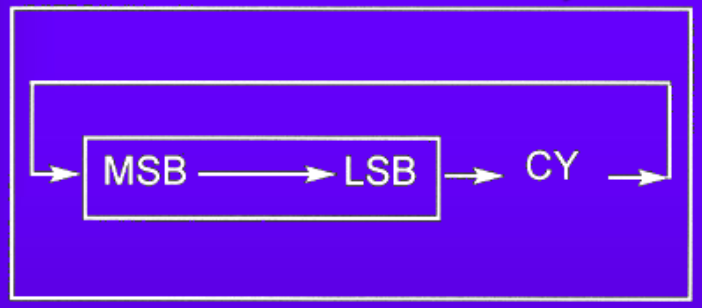

```
MOV   A,#36H      ;A=0011  0110
RR    A           ;A=0001  1011
RR    A           ;A=1000  1101
RR    A           ;A=1100  0110
RR    A           ;A=0110  0011


RL    A           ;rotate left A
```

```
RL      A       ;rotate left A
```


```
MOV   A,#72H        ;A=0111  0010
RL    A             ;A=1110  0100
RL    A             ;A=1100  1001
```

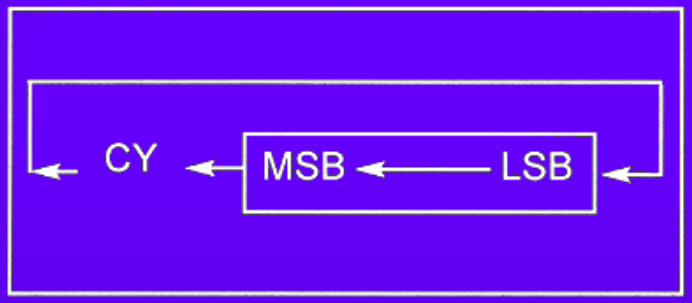
Rotating through the carry

RRC A ;rotate right through carry



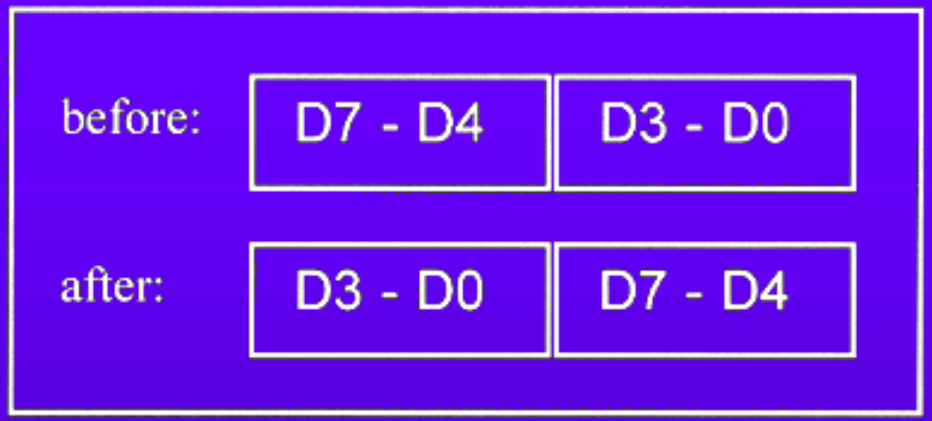

```
CLR   C          ;make CY=0
MOV   A,#26H     ;A=0010 0110
RRC   A          ;A=0001 0011  CY=0
RRC   A          ;A=0000 1001   CY=1
RRC   A          ;A=1000 0100   CY=1


RLC   A          ;rotate left through carry
```

```
RLC     A               ;rotate left through carry
```




```
SETB  C               ;make CY=1
MOV   A,#15H          ;A=0001  0101
RLC   A               ;A=0010  1010  CY=0
RLC   A               ;A=0101  0110  CY=0
RLC   A               ;A=1010  1100  CY=0
RLC   A               ;A=0101  1000  CY=1
```

SWAP A


before:
D7 - D4
D3 - D0
after:
D3 - D0
D7 - D4


before:
0111
0010
after:
0010
0111

Example 7-12

(a) Find the contents of register A in the following code.
(b) In the absence of a SWAP instruction, how would you exchange the nibbles? Write a simple program to show the process.

Solution:

(a)

```
MOV   A,#72H      ;A = 72H
SWAP  A           ;A = 27H
```

(b)

```
MOV   A,#72H      ;A=0111 0010
RL    A           ;A=1110 0100
RL    A           ;A=1100 1001
RL    A           ;A=1001 0011
RL    A           ;A=0010 0111
```

Example 7-13

Write a program that finds the number of 1s in a given byte.

Solution:

```
        MOV   R1,#0      ;R1 keeps the number of 1s
        MOV   R7,#8      ;counter = 08 rotate 8 times
        MOV   A,#97H     ;find the number of 1s in 97H
AGAIN:  RLC   A          ;rotate it through the CY once
        JNC   NEXT       ;check for CY
        INC   R1         ;if CY=1 then add one to count
NEXT:   DJNZ  R7,AGAIN   ;go through this 8 times
```

```
RRC     A           ;first bit to carry
MOV     P1.3,C      ;output carry as data bit
RRC     A           ;second bit to carry
MOV     P1.3,C      ;output carry as data bit
RRC     A           ;third bit to carry
MOV     P1.3,C      ;output carry as data bit
…..
```

BCD AND ASCII APPLICATION PROGRAM

Table 7-2. ASCII Code for Digits 0 - 9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Packed BCD to ASCII conversion

Packed BCD	**Unpacked BCD**	**ASCII**
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

ASCII to packed BCD conversion

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

```
MOV     A,#'4'      ;A=34H, hex for ASCII char 4
MOV     R1,#'7'     ;R1=37H, hex for ASCII char 7
ANL     A,#0FH      ;mask upper nibble (A=04)
ANL     R1,#0FH     ;mask upper nibble (R1=07)
SWAP    A           ;A=40H
ORL     A,R1        ;A=47H, packed BCD
```

JUMP, LOOP and CALL Instructions

Outlines

- Loop instructions
- Conditional jump instructions
- Conditions determining conditional jump
- Unconditional long & short jumps
- Calculate target addresses for jumps
- Subroutines
- Using stack in subroutines
- Crystal frequency vs. machine cycle
- Code programs to generate time delay

Looping in 8051

- Repeating a sequence of instructions a certain number of times is called a loop.

Looping

Example 3-1

Write a program to
(a) clear ACC, then
(b) add 3 to the accumulator ten times.

Solution:

```
;This program adds value 3 to the ACC ten times

          MOV   A,#0        ;A=0, clear ACC
          MOV   R2,#10      ;load counter R2=10
AGAIN:    ADD   A,#03       ;add 03 to ACC
          DJNZ  R2,AGAIN    ;repeat until R2=0(10 times)
          MOV   R5,A        ;save A in R5
```

Example 3.2 :-WAP to add 1st 10 natural numbers

1st 10 natural numbers:-1,2,3,4,5,6,7,8,9,10

```
        MOV A,#0    ;A=0,clear ACC
        MOV R2,#10
        MOV R0,#0
AGAIN :INC R0
        ADD A,R0
        DJNZ R2,AGAIN
        MOV 46H,A
```

Loop inside a Loop (Nested Loop)

Example 3-3

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times.

Solution:

Since 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use R2 and R3 for the count.

```
        MOV   A,#55H      ;A=55H
        MOV   R3,#10      ;R3=10, the outer loop count
NEXT:   MOV   R2,#70      ;R2=70, the inner loop count
AGAIN:  CPL   A           ;complement A register
        DJNZ  R2,AGAIN    ;repeat it 70 times (inner loop)
        DJNZ  R3,NEXT
```

In this program, R2 is used to keep the inner loop count. In the instruction "DJNZ R2,AGAIN", whenever R2 becomes 0 it falls through and "DJNZ R3,NEXT" is executed. This instruction forces the CPU to load R2 with the count 70 and the inner loop starts again. This process will continue until R3 becomes zero and the outer loop is finished.

8051 Conditional Jump Instructions

Table 3-1: 8051 Conditional Jump Instructions

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and jump if A ≠ 0
CJNE A,byte	Jump if A ≠ byte
CJNE reg,#data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

Conditional Jump Example

Example 3-4

Write a program to determine if R5 contains the value 0. If so, put 55H in it.

Solution:

```
            MOV   A,R5        ;copy R5 to A
            JNZ   NEXT        ;jump if A is not zero
            MOV  R5,#55H
NEXT:       ...
```

Conditional Jump Example

Example 3-5

Find the sum of the values 79H, F5H, and E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

Solution:

```
        MOV   A,#0        ;clear A(A=0)
        MOV   R5,A        ;clear R5
        ADD   A,#79H      ;A=0+79H=79H
        JNC   N_1         ;if no carry, add next number
        INC   R5          ;if CY=1, increment R5
N_1:    ADD   A,#0F5H     ;A=79+F5=6E and CY=1
        JNC   N_2         ;jump if CY=0
        INC   R5          ;If CY=1 then increment R5(R5=1)
N_2:    ADD   A,#0E2H     ;A=6E+E2=50 and CY=1
        JNC   OVER        ;jump if CY=0
        INC   R5          ;if CY=1, increment 5
OVER:   MOV   R0,A        ;Now R0=50H, and R5=02
```

Unconditional Jump Instructions

- All conditional jumps are short jumps
  - Target address within -128 to +127 of PC
- LJMP (long jump): 3-byte instruction
  - 2-byte target address: 0000 to FFFFH
  - Original 8051 has only 4KB on-chip ROM
- SJMP (short jump): 2-byte instruction
  - 1-byte relative address: -128 to +127
  - Relative address: 00 to FFH

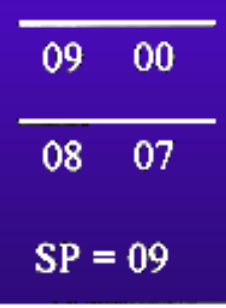
Call Instructions

- LCALL (long call): 3-byte instruction
  - 2-byte address
  - To call subroutines located anywhere within 64K-byte range
- ACALL (absolute call): 2-byte instruction
  - 11-bit address
  - Target address within 2K-byte range

CALL Instruction & Role of Stack

```
011   0300                    ORG 300H
012   0300          DELAY:
013   0300 7DFF               MOV  R5,#0FFH  ;R5=255
014   0302 DDFE   AGAIN:      DJNZ R5,AGAIN  ;stay here
015   0304 22                 RET            ;return to caller
016   0305                    END            ;end of asm file
```

When the first LCALL is executed, the address of the instruction "MOV A,#0AAH" is saved on the stack. Notice that the low byte goes first and the high byte is last. The last instruction of the called subroutine must be a RET instruction which directs the CPU to POP the top bytes of the stack into the PC and resume executing at address 07. The diagram shows the stack frame after the first LCALL.



Calling Subroutines

```
;MAIN program calling subroutines
            ORG  0
MAIN:       LCALL      SUBR_1
            LCALL      SUBR_2
            LCALL      SUBR_3

HERE:       SJMP       HERE
;———————end of MAIN
;
SUBR_1:     ....
            ....
            RET
;———————end of subroutine 1
```

Figure 3-1. 8051 Assembly Main Program That Calls Subroutines

Calling Subroutines

```
;
SUBR_2:     ....
            ....
            RET
;-----------end of subroutine 2

SUBR_3:     ....
            ....
            RET
;-----------end of subroutine 3
            END          ;end of the asm file
```

Figure 3-1. 8051 Assembly Main Program That Calls Subroutines

ACALL (absolute call)

Example 3-11

A developer is using the Atmel AT89C1051 microcontroller chip for a product. This chip has only 1K bytes of on-chip flash ROM. Which of the instructions LCALL and ACALL is most useful in programming this chip?

Solution:
The ACALL instruction is more useful since it is a 2-byte instruction. It saves one byte each time the call instruction is used.

Time Delay Generation & Calculation

- 1 instruction = $n \times$ machine cycle
- 1 machine cycle = 12 clock cycles

Example 3-13

The following shows crystal frequency for three different 8051-based systems. Find the period of the machine cycle in each case.
(a) 11.0592 MHz (b) 16 MHz (c) 20 MHz

Solution:

(a)11.0592/12 = 921.6 kHz; machine cycle is 1/921.6 kHz = 1.085 μs (microsecond)
(b) 16 MHz/12 = 1.333 MHz; machine cycle (MC) = 1/.1.333 MHz = 0.75 μs
(c) 20 MHz/12 = 1.66 MHz; MC = 1/1.66 MHz = 0.60 μs

Machine cycle of 8051

Instruction	8051
MOV Rx,#value	1
DEC Rx	1
DJNZ	2
LJMP	2
SJMP	2
NOP	1
MUL AB	4

Delay Calculation

Example 3-14

For an 8051 system of 11.0592 MHz, find how long it takes to execute each of the following instructions.

(a) `MOV R3,#55` (b) `DEC R3` (c) `DJNZ R2,target`
(d) `LJMP` (e) `SJMP` (f) `NOP` (no operation)
(g) `MUL AB`

Solution:

The machine cycle for a system of 11.0592 MHz is 1.085 µs as shown in Example 3-13. Table A-1 in Appendix A shows machine cycles for each of the above instructions. Therefore, we have:

Instruction	*Machine cycles*	*Time to execute*
(a) `MOV R3,#55`	1	1×1.085 µs = 1.085 µs
(b) `DEC R3`	1	1×1.085 µs = 1.085 µs
(c) `DJNZ R2,target`	2	2×1.085 µs = 2.17 µs
(d) `LJMP`	2	2×1.085 µs = 2.17 µs
(e) `SJMP`	2	2×1.085 µs = 2.17 µs
(f) `NOP`	1	1×1.085 µs = 1.085 µs
(g) `MUL AB`	4	4×1.085 µs = 4.34 µs

Delay Calculation Example

Example 3-15

Find the size of the delay in the following program, if the crystal frequency is 11.0592 MHz.

```
            MOV   A,#55H
AGAIN:      MOV   P1,A
            ACALL DELAY
            CPL   A
            SJMP AGAIN
;----Time delay
DELAY:      MOV   R3,#200
HERE:       DJNZ R3,HERE
            RET
```

Delay Calculation Example

Solution:

From Table A-1 in Appendix A, we have the following machine cycles for each instruction of the DELAY subroutine.

```
                                  Machine Cycle
DELAY:      MOV   R3,#200             1
HERE:       DJNZ  R3,HERE             2
            RET                       1
```

Therefore, we have a time delay of $[(200 \times 2) + 1 + 1] \times 1.085\ \mu s = 436.17\ \mu s$.

Increasing Delay Using NOP

Example 3-16

Find the time delay for the following subroutine, assuming a crystal frequency of 11.0592 MHz.

```
                                  Machine Cycle
DELAY:    MOV   R3,#250             1

HERE:     NOP                       1
          NOP                       1
          NOP                       1
          NOP                       1
          DJNZ  R3,HERE             2

          RET                       1
```

Solution:

The time delay inside the HERE loop is [250 (1+1+1+1+2)] × 1.085 μs = 1500 × 1.085 μs = 1627.5 μs. Adding the two instructions outside the loop we have 1627.5 μs + 2 × 1.085 μ s = 1629.67 μs.

Generate delay of 5 ms

```
DELAY: MOV R2,#10          1
  HERE: MOV R3,#255        1
AGAIN: DJNZ R3,AGAIN       2
       DJNZ R2,HERE        2
        RET
```

1 m/c cycle =11.0592 Hz

Again loop takes (2*255)*1.085 =553.35 us

Here loop repeat again loop 10 times

So delay = 553.35*10 =5.5335 ms =5 ms

- Number*M/c cycle*time period for 1 m/c cycle = delay

500000(0.5 sec) =number*2*1.085

Number = 230414(255*y)

y=230414/255 =900(255*z)

Z=900/255 =4

Generate delay of 0.5 sec

```
DELAY: MOV R2,#4              1
 HERE1: MOV R1,#255           1
 HERE2: MOV R0,#255           1
 HERE3: DJNZ R0,HERE3         2
        DJNZ R1,HERE2         2
        DJNZ R2,HERE1         2
        RET
```

DELAY = 4*255*255*2 MC*1.085 us =0.5 second

Chapter 6
Arithmetic Instructions and Programs

Outlines

- Range of numbers in 8051 unsigned data
- Addition & subtraction instructions for unsigned data
- BCD system of data representation
- Packed and unpacked BCD data
- Addition & subtraction on BCD data
- Range of numbers in 8051 signed data
- Signed data arithmetic instructions
- Carry & overflow problems & corrections

Addition of Unsigned Numbers

- ADD A, source ; A = A + source

ADDC & Addition of 16-bit Numbers

	1	
	3C	E7
+	3B	8D
	78	74

Example 6-3

Write a program to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Place the sum in R7 and R6; R6 should have the lower byte.

Solution:

```
CLR   C          ;make CY=0
MOV   A,#0E7H    ;load the low byte now A=E7H
ADD   A,#8DH     ;add the low byte now A=74H and CY=1
MOV   R6,A       ;save the low byte of the sum in R6
MOV   A,#3CH     ;load the high byte
ADDC  A,#3BH     ;add with the carry
                 ;3B + 3C + 1 = 78(all in hex)
MOV   R7,A       ;save the high byte of the sum
```

BCD Number System

- Unpacked BCD: 1 byte
- Packed BCD: 4 bits

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure 6.1. BCD Code

Adding BCD Numbers & DA Instruction

```
MOV   A,#17H
ADD   A,#28H
```

```
MOV   A,#47H      ;A=47H first BCD operand
MOV   B,#25H      ;B=25 second BCD operand
ADD   A,B         ;hex (binary) addition (A=6CH)
DA    A           ;adjust for BCD addition (A=72H)
```

	HEX		***BCD***	
	29		0010 1001	
+	18	+	0001 1000	
	41		0100 0001	AC=1
+	6	+	0110	
	47		0100 0111	

Subtraction of Unsigned Numbers

- SUBB A, source ; A = A – source – CY
- SUBB when CY = 0
  - Take 2's complement of subtraend (source)
  - Add it to minuend
  - Invert carry

Example (Positive Result)

Example 6-5

Show the steps involved in the following.

```
CLR  C          ;make CY=0
MOV  A,#3FH     ;load 3FH into A (A=3FH)
MOV  R3,#23H    ;load 23H into R3(R3=23H)
SUBB A,R3       ;subtract A - R3, place result in A
```

Solution:

```
A =  3F   0011 1111      0011 1111
R3 = 23   0010 0011    + 1101 1101 (2's complement)
     1C                1 0001 1100
                       0 CF=0 (step 3)
```

The flags would be set as follows: CY = 0, AC = 0, and the programmer must look at the carry flag to determine if the result is positive or negative.

Example (Negative Result)

Example 6-6

Analyze the following program:

```
      CLR   C
      MOV   A,#4C       ;load A with value 4CH (A=4CH)
      SUBB  A,#6EH      ;subtract 6E from A
      JNC   NEXT        ;if CY=0 jump to NEXT target
      CPL   A           ;if CY=1 then take 1's complement
      INC   A           ;and increment to get 2's complement
NEXT:MOV    R1,A        ;save A in R1
```

Solution:

Following are the steps for "SUBB A,6EH":

```
      4C    0100 1100                    0100 1100
   -  6E    0110 1110       2's comp =   1001 0010
     -22                                 01101 1110
CY=1, the result is negative, in 2's complement.
```

SUBB When CY = 1

- For multibyte numbers

Example 6-7

Analyze the following program:

```
CLR   C                ;CY=0
MOV   A,#62H           ;A=62H
SUBB  A,#96H           ;62H-96H=CCH with CY=1
MOV   R7,A             ;save the result
MOV   A,#27H           ;A=27H
SUBB  A,#12H           ;27H-12H-1=14H
MOV   R6,A             ;save the result
```

Solution:

After the SUBB, A = 62H – 96H = CCH and the carry flag is set high indicating there is a borrow. Since CY = 1, when SUBB is executed the second time A = 27H – 12H – 1 =14H. Therefore, we have 2762H – 1296H = 14CCH.

Multiplication of Unsigned Numbers

- MUL AB ; A × B, place 16-bit result in B and A

```
MOV     A,#25H      ;load 25H to reg. A
MOV     B,#65H      ;load 65H in reg. B
MUL     AB          ;25H * 65H = E99 where
                    ;B = 0EH and A = 99H
```

Table 6-1:Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand 1	Operand 2	Result
byte × byte	A	B	A=low byte, B=high byte

Division of Unsigned Numbers

- DIV AB ; divide A by B

```
MOV   A,#95H    ;load 95 into A
MOV   B,#10H    ;load 10 into B
DIV   AB        ;now A = 09 (quotient) and
                ;B = 05 (remainder)
```

Table 6-2:Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
byte / byte	A	B	A	B

Example (1 of 2)

Example 6-8

(a) Write a program to get hex data in the range of 00 - FFH from port 1 and convert it to decimal. Save the digits in R7, R6, and R5, where the least significant digit is in R7.
(b) Analyze the program, assuming that P1 has a value of FDH for data.

Solution:

(a)

```
MOV   A,#0FFH
MOV   P1,A        ;make P1 an input port
MOV   A,P1        ;read data from P1
MOV   B,#10       ;B=0A hex (10 dec)
DIV   AB          ;divide by 10
MOV   R7,B        ;save lower digit
MOV   B,#10       ;
DIV   AB          ;divide by 10 once more
MOV   R6,B        ;save the next digit
MOV   R5,A        ;save the last digit
```

Example (2 of 2)

(b)
To convert a binary (hex) value to decimal, we divide it by 10 repeatedly until the quotient is less than 10. After each division the remainder is saved. In the case of an 8-bit binary such as FDH we have 253 decimal as shown below (all in hex).

```
            Q          R
FD/0A=      19         3  (low digit)
19/0A=      2          5  (middle digit)
                       2  (high digit)
```

Therefore, we have FDH = 253. In order to display this data it must be converted to ASCII, which is described in the next chapter.

Signed 8-bit Operands

- Covert to 2's complement
  - Write magnitude of number in 8-bit binary (no sign)
  - Invert each bit
  - Add 1 to it

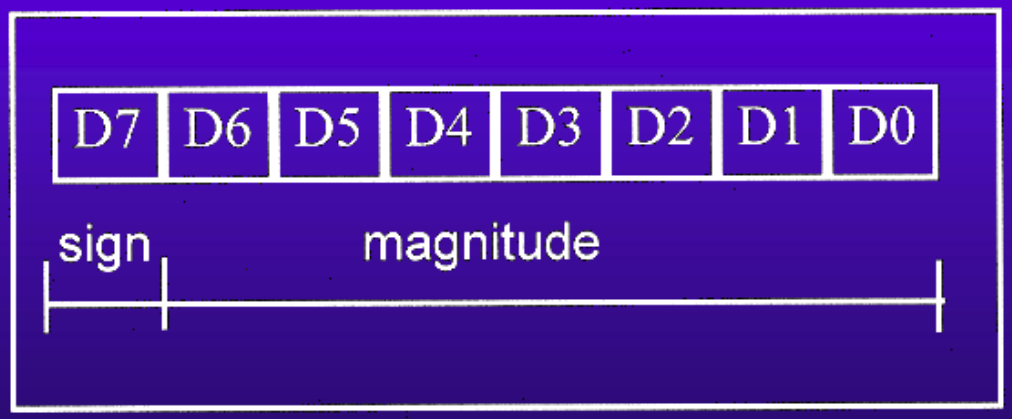


Figure 6-2. 8-Bit Signed Operand

Example

Example 6-9

Show how the 8051 would represent -5.

Solution:

Observe the following steps.

```
1.    0000 0101       5 in 8-bit binary
2.    1111 1010       invert each bit
3     1111 1011       add 1 (which becomes FB in hex)
```

Therefore -5 = FBH, the signed number representation in 2's complement for -5.

Example

Example 6-10

Show how the 8051 would represent –34H.

Solution:

Observe the following steps.

```
1.    0011 0100       34H given in binary
2.    1100 1011       invert each bit
3     1100 1100       add 1 (which is CC in hex)
```

Therefore –34 = CCH, the signed number representation in 2's complement for –34H.

Example

Example 6-11

Show how the 8051 would represent –128.

Solution:

Observe the following steps.

```
1.    1000 0000       128 in 8-bit binary
2.    0111 1111       invert each bit
3     1000 0000       add 1 (which becomes 80 in hex)
```

Therefore –128 = 80H, the signed number representation in 2's complement for –128.

Byte-sized Signed Numbers Ranges

Decimal	*Binary*	*Hex*
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
….	……	..
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
…	……	...
+127	0111 1111	7F

Overflow in Signed Number Operations

Example 6-12

Examine the following code and analyze the result.

```
MOV   A,#+96          ;A=0110 0000(A=60H)
MOV   R1,#+70         ;R1=0100 0110(R1=46H)
ADD   A,R1            ;A=1010 0110
                      ;A=A6H= -90 decimal, INVALID!!)
```

Solution:

```
    +96  0110 0000
+   +70  0100 0110
+  166   1010 0110   and OV=1
```

According to the CPU, the result is -90, which is wrong. The CPU sets OV = 1 to indicate the overflow.

When Is the OV Flag Set?

- Either: there is a carry from D6 to D7 but no carry out of D7 (CY = 0)
- Or: there is a carry from D7 out (CY = 1) but no carry from D6 to D7

Example

Example 6-13

Observe the following, noting the role of the OV flag.

```
     MOV   A,#-128     ;A=1000 0000(A=80H)
     MOV   R4,#-2      ;R4=1111 1110(R4=FEH)
     ADD   A,R4        ;A=0111 1110 (A=7EH=+126, invalid)
```

Solution:

```
  -128        1000 0000
+ - 2         1111 1110
- 130         0111 1110   and OV=1
```

According to the CPU, the result is +126, which is wrong (OV = 1).

Example

Example 6-14

Observe the following, noting the OV flag.

```
MOV   A,#-2              ;A=1111 1110 (A=FEH)
MOV   R1,#-5             ;R1=1111 1011(R1=FBH)
ADD   A,R1               ;A=1111 1001 (A=F9H=-7,correct,OV=0)
```

Solution:

```
    -2   1111 1110
+   -5   1111 1011
  - 7    1111 1001   and OV = 0
```

According to the CPU, the result is -7, which is correct (OV = 0).

Example

Example 6-15

Examine the following, noting the role of OV.

```
MOV   A,#+7              ;A=0000 0111 (A=07H)
MOV   R1,#+18            ;R1=0001 0010(R1=12H)
ADD   A,R1               ;A=0001 1001 (A=19H=+25, correct,OV=0)
```

Solution:

```
        7  0000 0111
  +    18  0001 0010
       25  0001 1001   and OV = 0
```

According to the CPU, this is +25, which is correct (OV = 0)

Chapter 8
SINGLE-BIT INSTRUCTIONS AND PROGRAMMING

Outlines

- List the 8051 Assembly language instructions for bit manipulation
- Code 8051 instructions for bit manipulation of ports
- Explain which 8051 registers are bit-addressable
- Describe which portions of the 8051 RAM are bit-addressable
- Discuss bit manipulation of the carry flag
- Describe the carry flag bit-related instructions of the 8051

Single-bit instructions

Table 8-1: Single-Bit Instructions

Instruction		Function
SETB	bit	Set the bit (bit = 1)
CLR	bit	Clear the bit (bit = 0)
CPL	bit	Complement the bit (bit = NOT bit)
JB	bit,target	Jump to target if bit = 1 (jump if bit)
JNB	bit,target	Jump to target if bit = 0 (jump if no bit)
JBC	bit,target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

I/O ports and bit-addressability

The 8051 has four I/O ports, each of which is 8 bits

Example 8-1

Write the following programs.

(a) Create a square wave of 50% duty cycle on bit 0 of port 1.

(b) Create a square wave of 66% duty cycle on bit 3 of port 1.

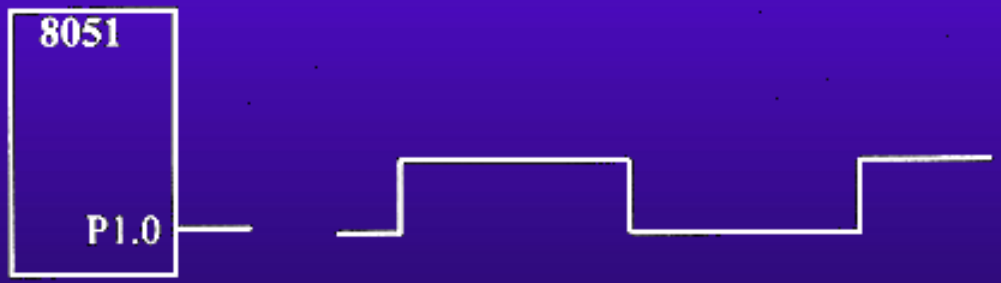
Solution:

(a) The 50% duty cycle means that the "on" and "off" state (or the high and low portion of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

```
HERE:   SETB  P1.0     ;set to high bit 0 of port 1
        LCALL DELAY    ;call the delay subroutine
        CLR   P1.0     ;P1.0=0
        LCALL DELAY
        SJMP  HERE     ;keep doing it
```

Another way to write the above program is:

```
HERE:   CPL   P1.0     ;complement bit 0 of port 1
        LCALL DELAY    ;call the delay subroutine
        SJMP  HERE     ;keep doing it
```



(b) The 66% duty cycle means the "on" state is twice the "off" state.

```
BACK:    SETB P1.3       ;set port 1 bit 3 high
         LCALL DELAY     ;call the delay subroutine
         LCALL DELAY     ;call the delay subroutine again
         CLR  P1.3       ;clear bit 2 of port 1(P1.3=low)
         LCALL DELAY     ;call the delay subroutine
         SJMP  BACK      ;keep doing it
```

Table 8-2: Single-Bit Addressability of Ports

P0	P1	P2	P3	Port's Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Example 8-2

For each of the following instructions, state which bit of which SFR will be affected. Use Figure 8-1.

(a) `SETB 86H` (b) `CLR 87H` (c) `SETB 92H`
(d) `SETB 0A7H` (e) `CLR 0F2H` (f) `SETB 0E7H`

Solution:

(a) `SETB 86H` is for `SETB P0.6`
(b) `CLR 87H` is for `CLR P0.7`
(c) `SETB 92H` is for `SETB P1.2`
(d) `SETB 0A7H` is for `SETB P2.7`
(e) `CLR 0F2H` is for `CLR D2 of register B`
(f) `SETB 0E7H` is for `SETB ACC.7 (D7 of register A)`

Checking an input bit

JNB (jump if no bit) ; JB (jump if bit = 1)

Example 8-3

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
HERE:JNB   P2.3,HERE        ;keep monitoring for high
     SETB  P1.5             ;set bit P1.5=1
     CLR   P1.5             ;make high-to-low
```

Registers and bit-addressability

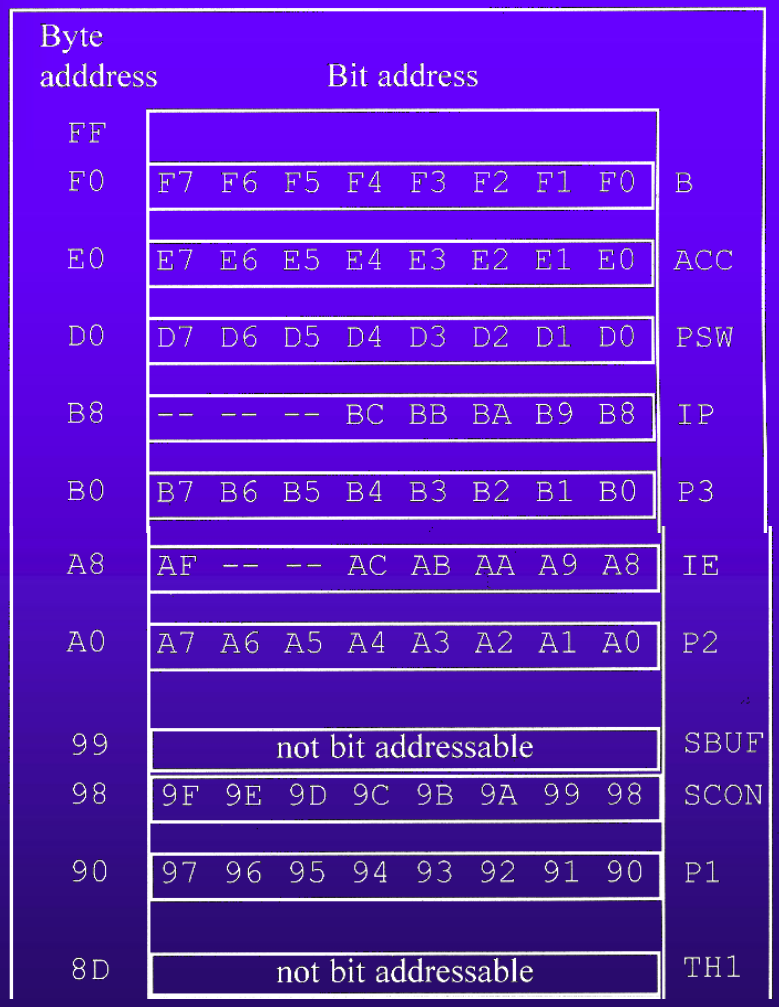


Figure 8-1. SFR RAM Address (Byte and Bit)

Example 8-4

Write a program to see if the accumulator contains an even number. If so, divide it by 2. If not, make it even and then divide it by 2.

Solution:

```
        MOV   B,#2          ;B=2
        JNB   ACC.0,YES ; is D0 of reg A 0?, if so jump
        INC   A             ;it is odd, make it even
YES:    DIV   AB            ;A/B
```

Example 8-5

Write a program to see if bits 0 and 5 of register B are 1. If they are not, make them so and save it in R0.

Solution:

```
        JNB   0F0H,NEXT_1     ;jump if B.0 is low
        SETB  0F0H            ;make bit B.0 high
NEXT_1: JNB   0F5H,NEXT_2     ;jump if B.5 is low
        SETB  0F5H            ;make B.5 high
NEXT_2: MOV   R0,B            ;save register B
```

CY	AC	--	RS1	RS0	OV	--	P

RS1	RS0	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Figure 8-2. Bits of the PSW Register

Example 8-6

Write a program to save the accumulator in R7 of bank 2.

Solution:

```
CLR   PSW.3
SETB  PSW.4
MOV   R7,A
```

Example 8-7

While there are instructions such as JNC and JC to check the carry flag bit (CY), there are no such instructions for the overflow flag bit (OV). How would you write code to check OV?

Solution:

The OV flag is PSW.2 of the PSW register. PSW is a bit-addressable register; therefore, we can use the following instruction to check the OV flag.

```
JB    PSW.2,TARGET    ;jump if OV=1
```

Bit-addressable RAM

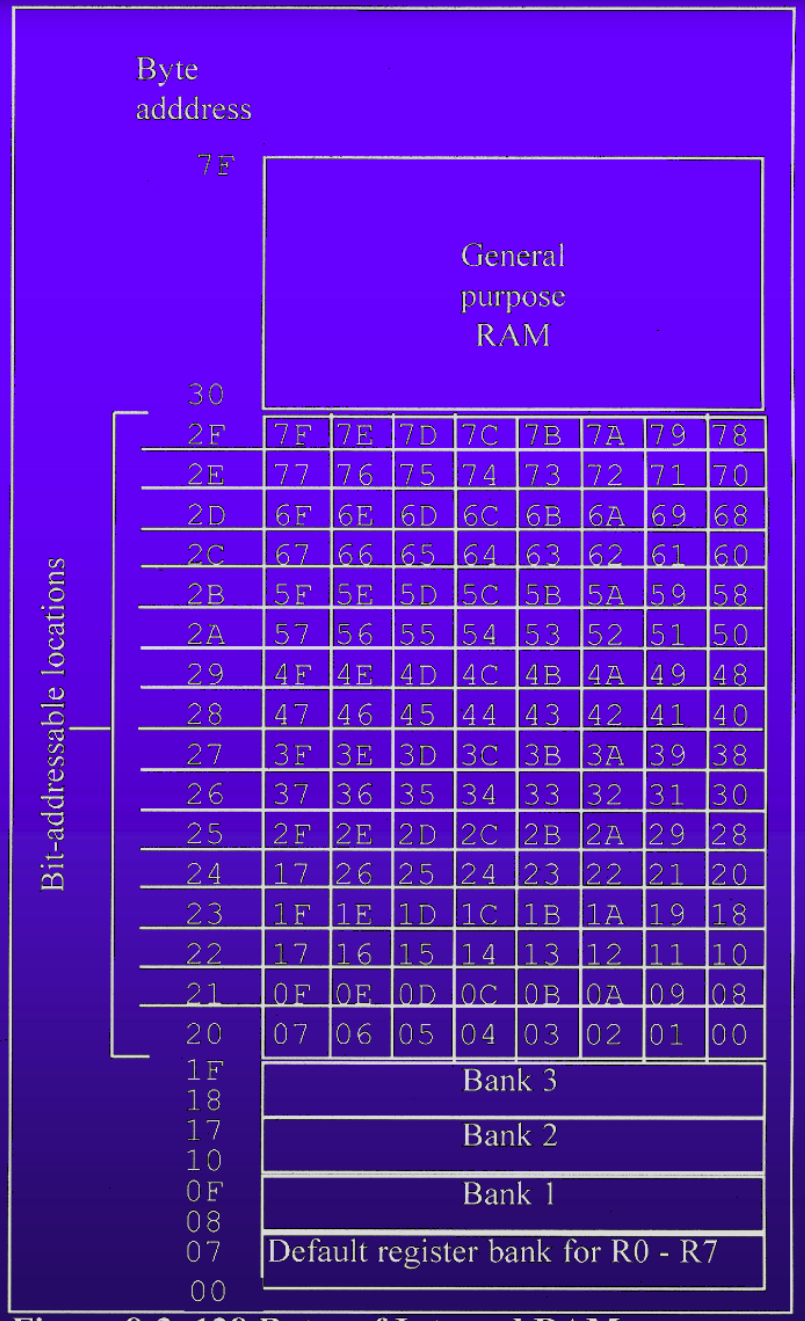


Figure 8-3. 128 Bytes of Internal RAM

Example 8-8

Find out to which byte each of the following bits belongs. Give the address of the RAM byte in hex.

```
(a) SETB 42H ;set bit 42H to 1
(b) CLR 67H ;clear bit 67
(c) CLR 0FH ;clear bit 0FH
(d) SETB 28H ;set bit 28H to 1
(e) CLR 12 ;clear bit 12 (decimal)
(f) SETB 05
```

Solution:

(a) RAM bit address of 42H belongs to D2 of RAM location 28H
(b) RAM bit address of 67H belongs to D7 of RAM location 2CH
(c) RAM bit address of 0FH belongs to D7 of RAM location 21H
(d) RAM bit address of 28H belongs to D0 of RAM location 25H
(e) RAM bit address of 12 belongs to D4 of RAM location 21H
(f) RAM bit address of 05 belongs to D5 of RAM location 20H

Example 8-9

The status of bits P1.2 and P1.3 of I/O port P1 must be saved before they are changed. Write a program to save the status of P1.2 in bit location 06 and the status of P1.3 in bit location 07.

Solution:

```
      CLR  06          ;clear bit address 06
      CLR  07          ;clear bit address 07
      JNB  P1.2,OVER   ;check bit P1.2,if 0 then jump
      SETB 06          ;if P1.2=1,set bit location 06 to 1
OVER: JNB  P1.3,NEXT   ;check bit P1.3,if 0 then jump
      SETB 07          ;if P1.3=1, set bit location 07 to 1
NEXT: ...
```

Single-bit operations with CY

Table 8-3: Carry Bit-Related Instructions

Instruction	Function
SETB C	make CY = 1
CLR C	clear carry bit (CY = 0)
CPL C	complement carry bit
MOV b,C	copy carry status to bit location (CY = b)
MOV C,b	copy bit location status to carry (b = CY)
JNC target	jump to target if CY = 0
JC target	jump to target if CY = 1
ANL C,bit	AND CY with bit and save it on CY
ANL C,/bit	AND CY with inverted bit and save it on CY
ORL C,bit	OR CY with bit and save it on CY
ORL C,/bit	OR CY with inverted bit and save it on CY

Example 8-10

Write a program to save the status of bits P1.2 and P1.3 on RAM bit locations 6 and 7, respectively.

Solution:

```
MOV C,P1.2    ;save status of P1.2 on CY
MOV 06,C      ;save carry in RAM bit location 06
MOV C,P1.3    ;save status of P1.3 on CY
MOV 07,C      ;save carry in RAM bit location 07
```

Example 8-11

Assume that RAM bit location 12H holds the status of whether there has been a phone call or not. If it is high, it means there has been a new call since it was checked the last time. Write a program to display "New Messages" on an LCD if bit RAM 12H is high. If it is low, the LCD should say "No New Messages".

Solution:

```
      MOV   C,12H             ;copy bit location 12H to carry
      JNC   NO                ;check to see if is high
      MOV   DPTR,#400H        ;yes, load address of message
      LCALL DISPLAY           ;display message (see Chap. 12)
      SJMP  EXIT              ;get out
NO:   MOV   DPTR,#420H        ;load the address of No message
      LCALL DISPLAY           ;display it
EXIT:                         ;exit
;-------------------data to be displayed on LCD
            ORG   400H
YES_MG:     DB    "New Messages"
            ORG   420H
NO_MG:      DB    "No New Messages"
```

Example 8-12

Assume that the bit P2.2 is used to control the outdoor light and bit P2.5 to control the light inside a building. Show how to turn on the outside light and turn off the inside one.

Solution:

```
STEB C          ;CY=1
ORL  C,P2.2     ;CY = P2.2 ORed with CY
MOV  P2.2,C     ;turn it "on" if not already "on"
CLR  C          ;CY=0
ANL  C,P2.5     ;CY=P2.5 ANDed with CY
MOV  P2.5,C     ;turn it off if not already off
```

READING INPUT PINS VS. PORT LATCH

In Reading a port:

1.Read the status of the input pin

2.Read the internal latch of the output port

Instructions for reading input port

Table 8-4:Instructions For Reading an Input Port

Mnemonics	Examples	Description
MOV A,PX	MOV A,P2	Bring into A the data at P2 pins
JNB PX.Y,..	JNB P2.1,TARGET	Jump if pin P2.1 is low
JB PX.Y,..	JB P1.3,TARGET	Jump if pin P1.3 is high
MOV C,PX.Y	MOV C,P2.4	Copy status of pin P2.4 to CY

Reading latch for output port

Table 8-5:Instructions Reading a Latch (Read-Modify-Write)

Mnemonics	Example
ANL Px	ANL P1,A
ORL Px	ORL P2,A
XRL Px	XRL P0,A
JBC PX.Y,TARGET	JBC P1.1,TARGET
CPL PX.Y	CPL P1.2
INC Px	INC P1
DEC Px	DEC P2
DJNZ PX.Y,TARGET	DJNZ P1,TARGET
MOV PX.Y,C	MOV P1.2,C
CLR PX.Y	CLR P2.3
SETB PX.Y	SETB P2.3